Trisha Lakhani

1BM19CS214

**SORT,REVERSE AND CONCATANATE**

#include <stdlib.h>

#include <string.h>

struct node

{

int sem;

struct node \*next;

};

struct node \*head= NULL;

struct node \*head2= NULL;

int c=0;

void Insert()

{

struct node \*newnode;

struct node \*temp;

int s;

printf("Enter integer : ");

scanf("%d",&s);

newnode=(struct node\*)malloc(sizeof(struct node));

newnode->sem =s;

if (head==NULL)

{

newnode->next=NULL;

head=newnode;

printf("first node of linked list created\n");

c++;

}

else

{

temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->next=NULL;

c++;

printf("Node created\n");

}

}

void Insert2()

{

struct node \*newnode;

struct node \*temp;

int s,y;

printf("enter elements to create list 2\n");

do

{

printf("Enter integer : \n");

scanf("%d",&s);

newnode=(struct node\*)malloc(sizeof(struct node));

newnode->sem =s;

if (head2==NULL)

{

newnode->next=NULL;

head2=newnode;

printf("first node of linked list created\n");

c++;

}

else

{

temp=head2;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->next=NULL;

c++;

printf("Node created\n");

}

printf("do u want to continue adding:0 or 1\n");

scanf("%d",&y);

}while(y!=0);

}

void bubbleSort()

{

int swapped, i;

struct node \*ptr1;

struct node \*lptr = NULL;

if (head == NULL)

return;

do

{

swapped = 0;

ptr1 = head;

while (ptr1->next != lptr)

{

if (ptr1->sem > ptr1->next->sem)

{

int temp = ptr1->sem;

ptr1->sem = ptr1->next->sem;

ptr1->next->sem = temp;

swapped = 1;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

}

while (swapped);

}

void reverse()

{

struct node\* prev = NULL;

struct node\* current = head;

struct node\* next ;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head= prev;

}

void concat()

{

struct node \*ptr;

if(head==NULL)

{

head=head2;

}

if(head2==NULL)

{

head2=head;

}

ptr=head;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=head2;

}

void display1()

{

struct node \*ptr;

ptr=head;

int i=1;

if(ptr==NULL)

{

printf("Linked list is empty!\n");

}

else

{

while(ptr!= NULL)

{

printf(" %d",ptr->sem);

i++;

ptr=ptr->next;

}

}

}

void display2()

{

struct node \*ptr;

ptr=head2;

int i=1;

if(ptr==NULL)

{

printf("Linked list is empty!\n");

}

else

{

while(ptr!= NULL)

{

printf(" %d",ptr->sem);

printf("\n");

i++;

ptr=ptr->next;

}

}

}

int main()

{

int choice,pos;

do

{

printf("\n1. Insert node \n2. sort node\n3. reverse node\n4.concat 2 lists \n5.exit\n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

Insert();

break;

case 2:

bubbleSort();

display1();

break;

case 3:

reverse();

display1();

break;

case 4:

Insert2();

concat();

display1();

break;

case 5:

break;

default:

printf("Wrong choice!\n");

break;

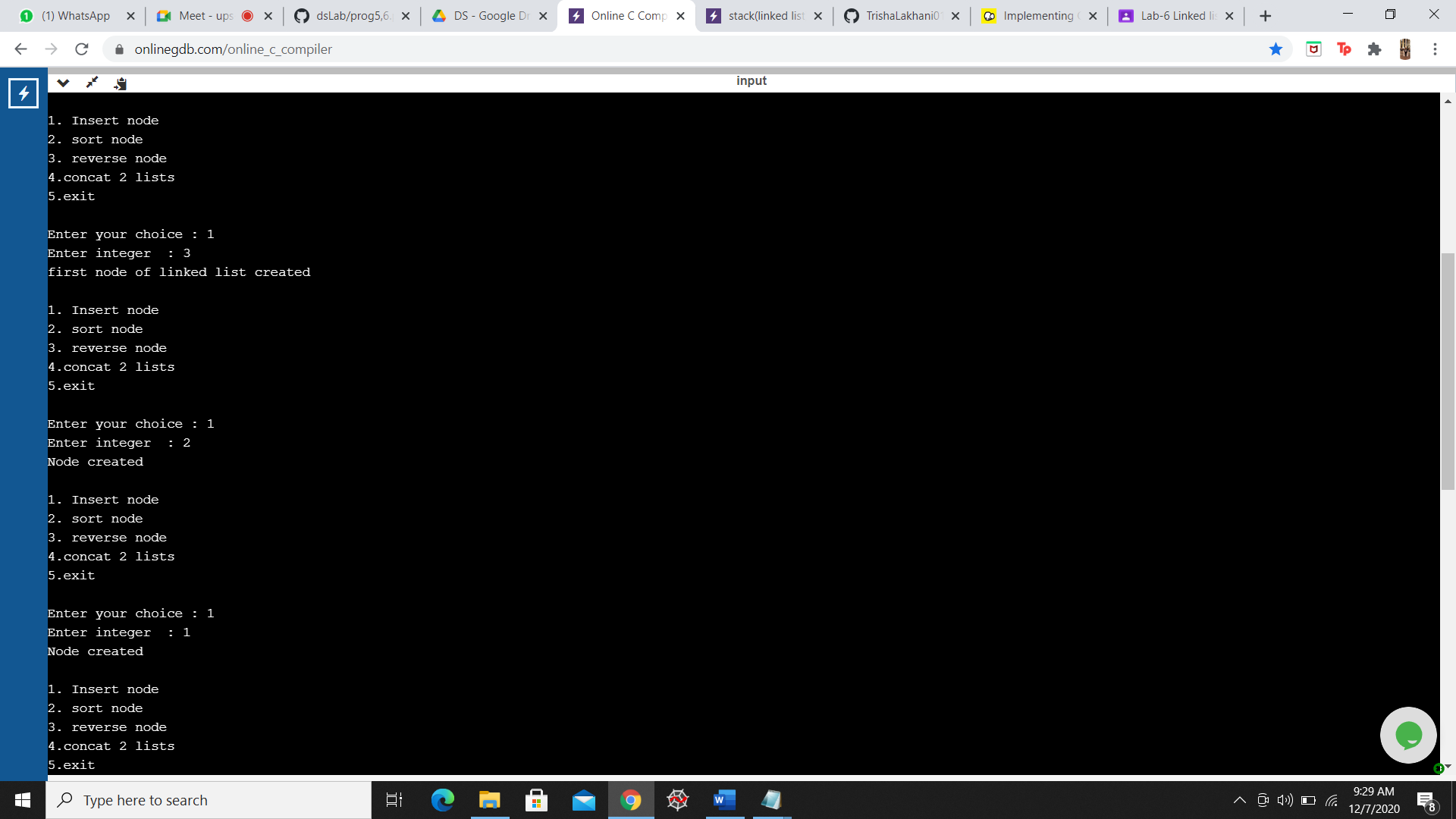
}

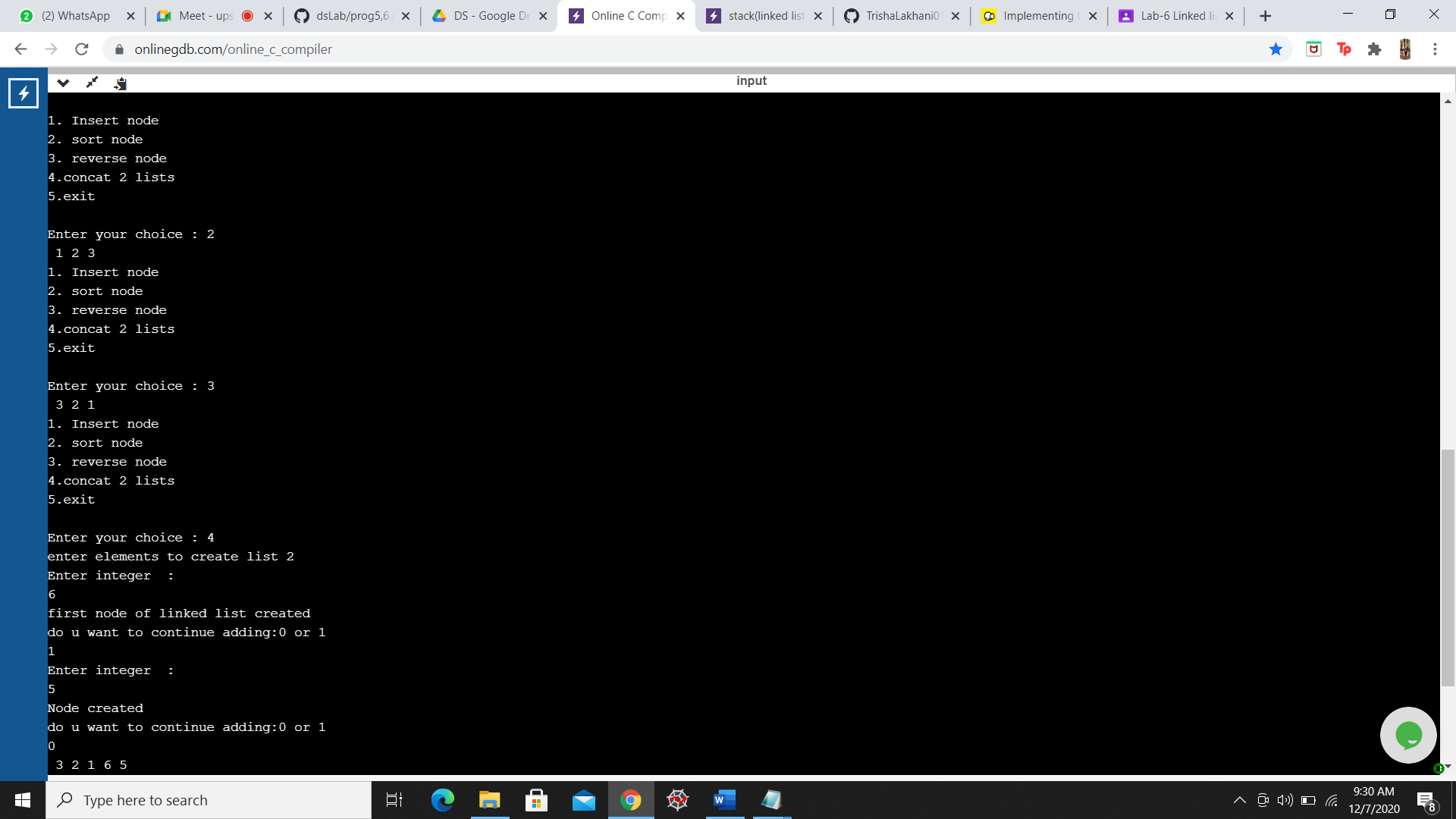
}while(choice!=5);

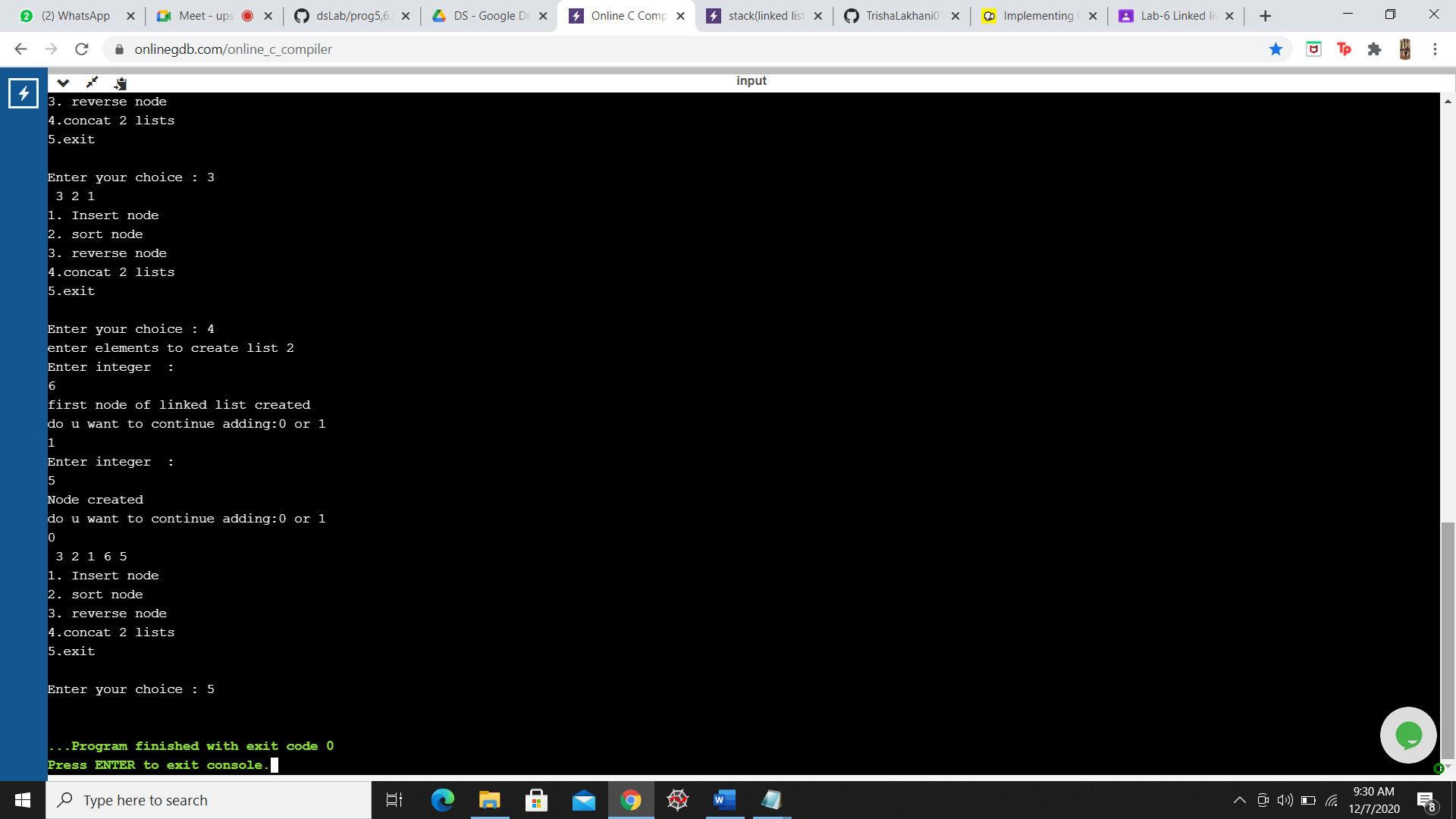
return 0;

}

**OUTPUT**

****

****

****

**STACKS USING LINKED LIST**

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

struct node

{

int val;

struct node \*next;

};

struct node \*head;

void main ()

{

int choice=0;

printf("\n\*\*\*\*\*\*\*\*\*Stack operations using linked list\*\*\*\*\*\*\*\*\*\n");

printf("\n----------------------------------------------\n");

while(choice != 4)

{

printf("\n\nChose one from the below options...\n");

printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

printf("\n Enter your choice \n");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("Exiting....");

break;

}

default:

{

printf("Please Enter valid choice ");

}

};

}

}

void push ()

{

int val;

struct node \*ptr = (struct node\*)malloc(sizeof(struct node));

if(ptr == NULL)

{

printf("not able to push the element");

}

else

{

printf("Enter the value");

scanf("%d",&val);

if(head==NULL)

{

ptr->val = val;

ptr -> next = NULL;

head=ptr;

}

else

{

ptr->val = val;

ptr->next = head;

head=ptr;

}

printf("Item pushed");

}

}

void pop()

{

int item;

struct node \*ptr;

if (head == NULL)

{

printf("Underflow");

}

else

{

item = head->val;

ptr = head;

head = head->next;

free(ptr);

printf("Item popped");

}

}

void display()

{

int i;

struct node \*ptr;

ptr=head;

if(ptr == NULL)

{

printf("Stack is empty\n");

}

else

{

printf("Printing Stack elements \n");

while(ptr!=NULL)

{

printf("%d\n",ptr->val);

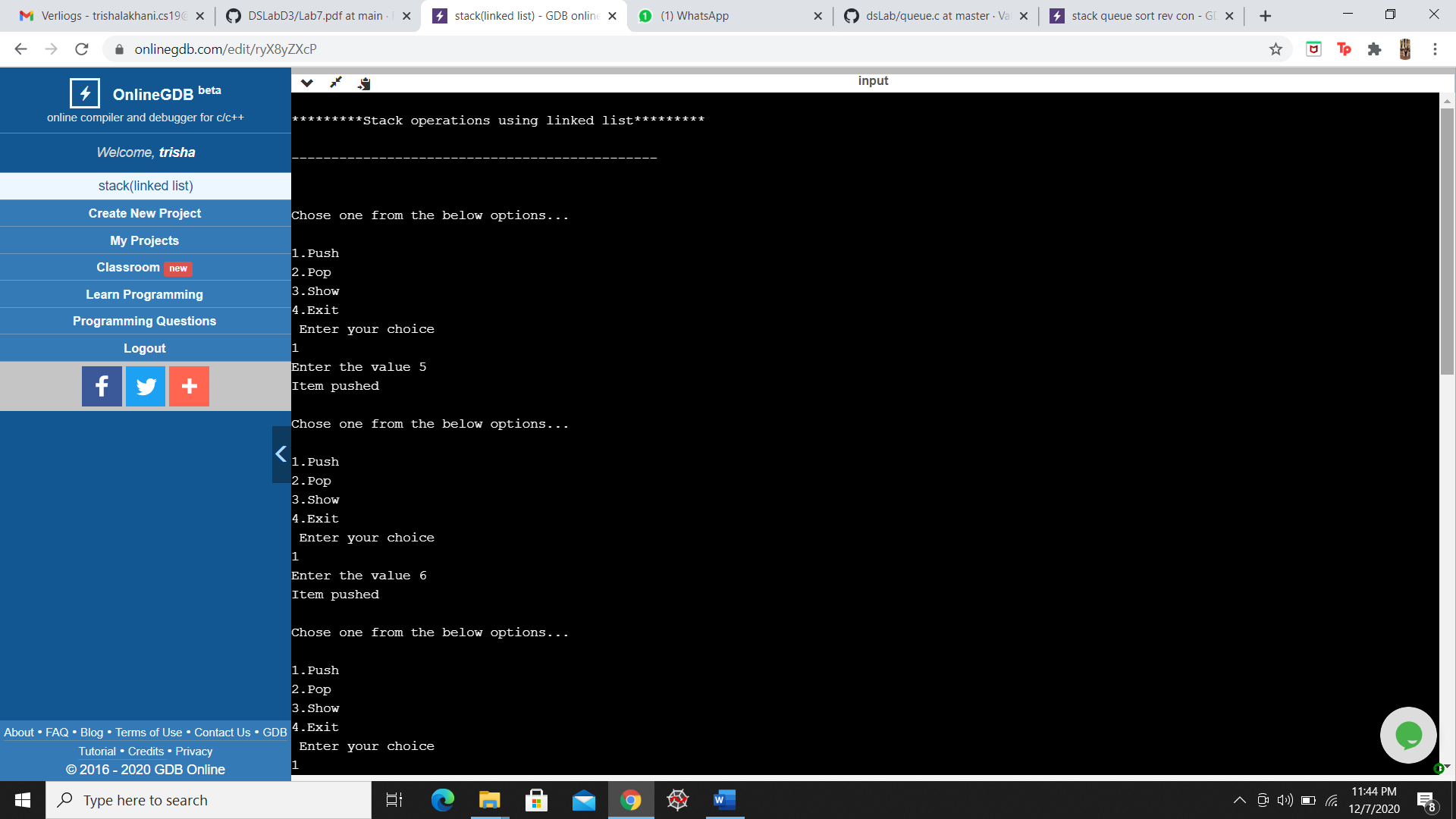
ptr = ptr->next;

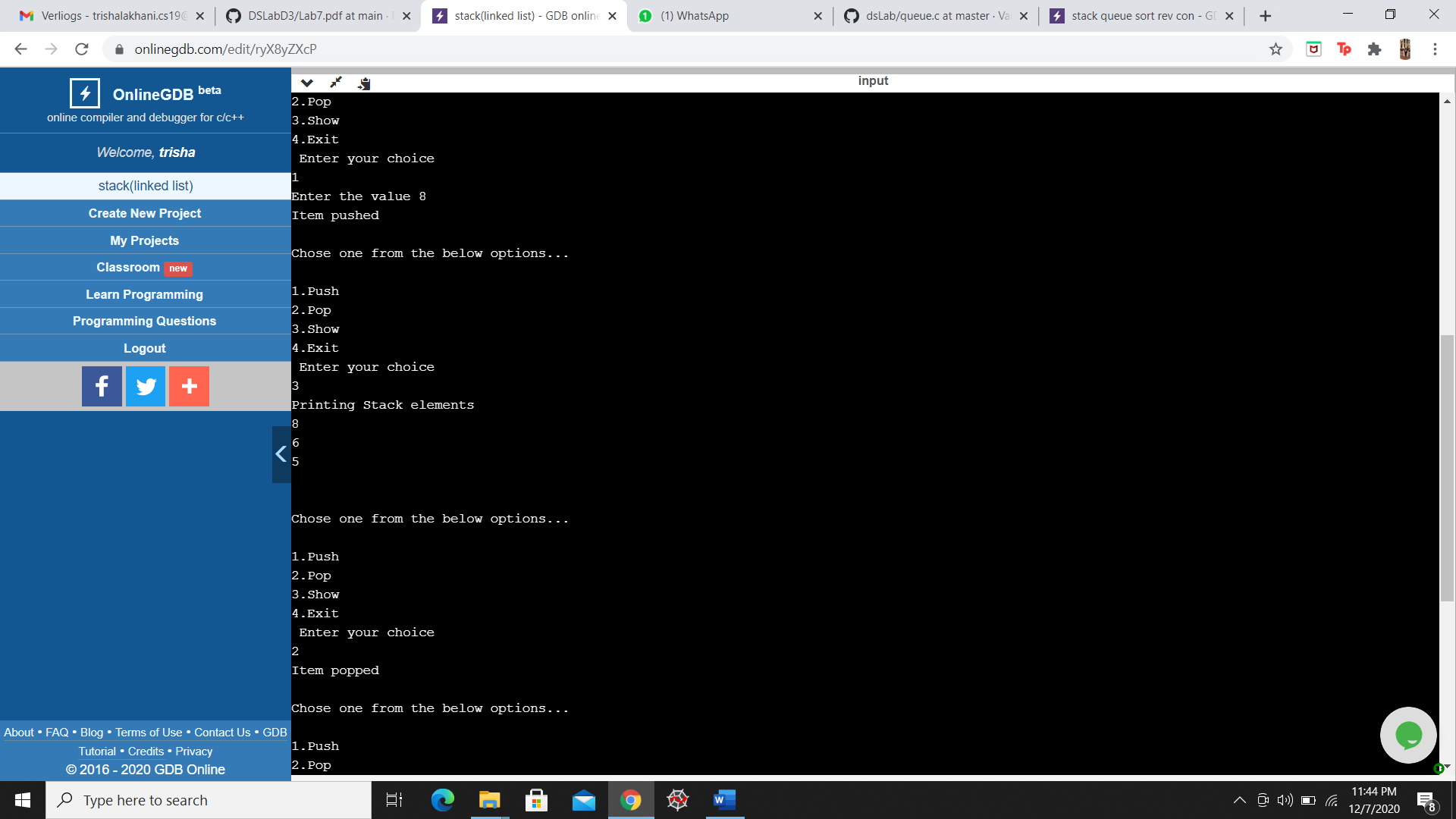
}

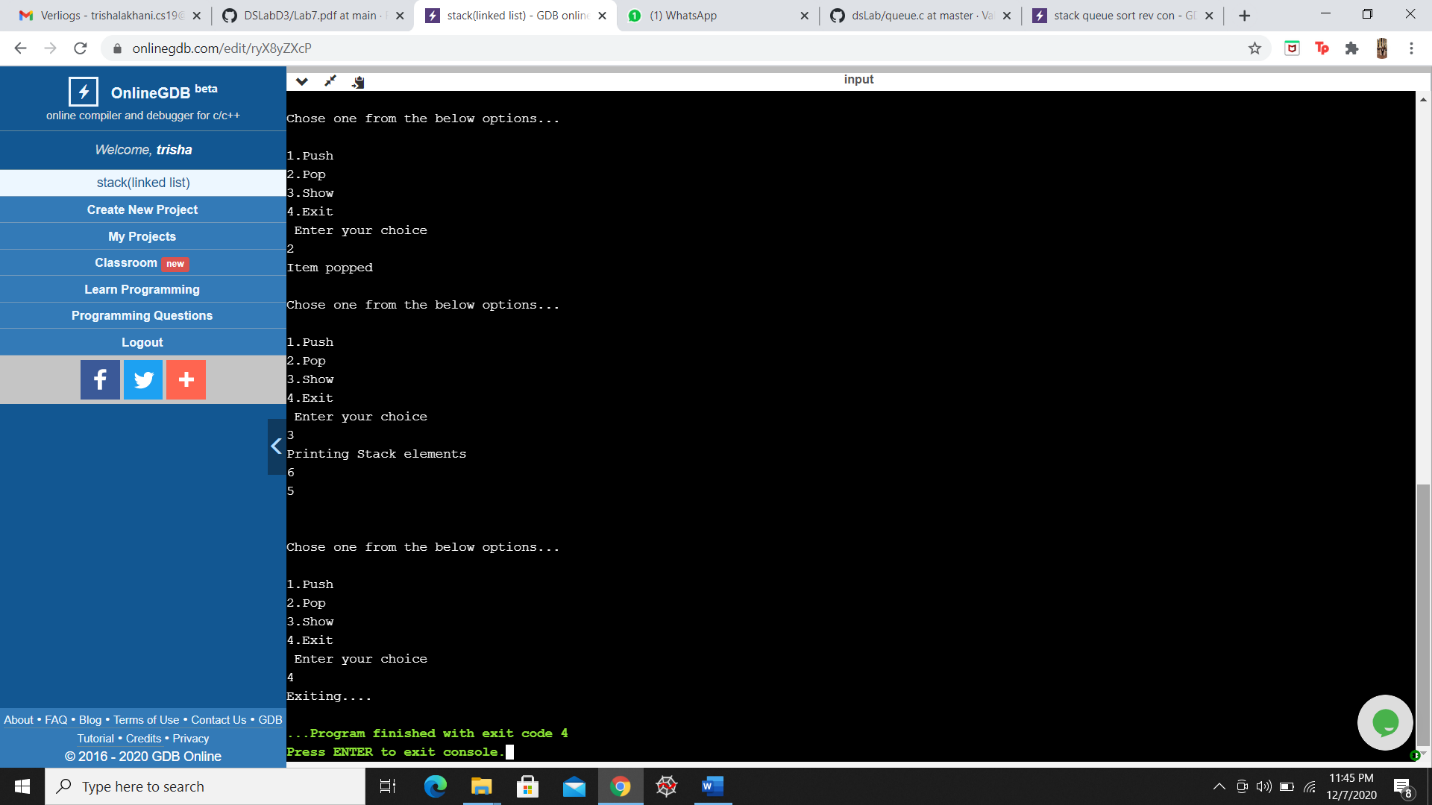
}

}

**OUTPUT**







**QUEUE USING LINKED LIST**

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int val;

struct node\* next;

}node;

node\* front = NULL;

node\* rear = NULL;

void enqueue(int input) {

node\* ptr = (node\*) malloc((sizeof(node)));

ptr->next = NULL;

ptr -> val = input;

if(front==NULL&&rear==NULL)

{

front = rear = ptr;

}

else{

rear->next = ptr;

rear = ptr;

}

printf("\n\nEnqueued\n\n");

}

void dequeue() {

if(front == NULL && rear == NULL){

printf("\n\nQueue is empty\n\n");

return;

}

if(front -> next == NULL)

{

free(front);

front = rear = NULL;

printf("\n\nDequeued\n\n");

return;

}

printf("\n\nDequeued element is %d",front->val);

front=front->next;

}

void display() {

if(front==NULL&&rear==NULL){

printf("\n\nQueue is empty\n\n");

return;

}

printf("Queue contains : ");

node\* ptr = front;

do

{

printf("%d ", ptr->val);

ptr = ptr->next;

} while(ptr!=NULL);

printf("\n\n");

}

int main() {

int choice, input;

while(1) {

printf("Enter 1 to enqueue\n");

printf("Enter 2 to dequeue\n");

printf("Enter 3 to display\n");

printf("Enter -1 to quit\n");

printf("Enter your choice : ");

scanf("%d", &choice);

if(choice == -1)

break;

switch(choice) {

case 1:

printf("Enter value to enqueue : ");

scanf("%d", &input);

enqueue(input);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

default:

printf("\n\nWrong Input\n\n");

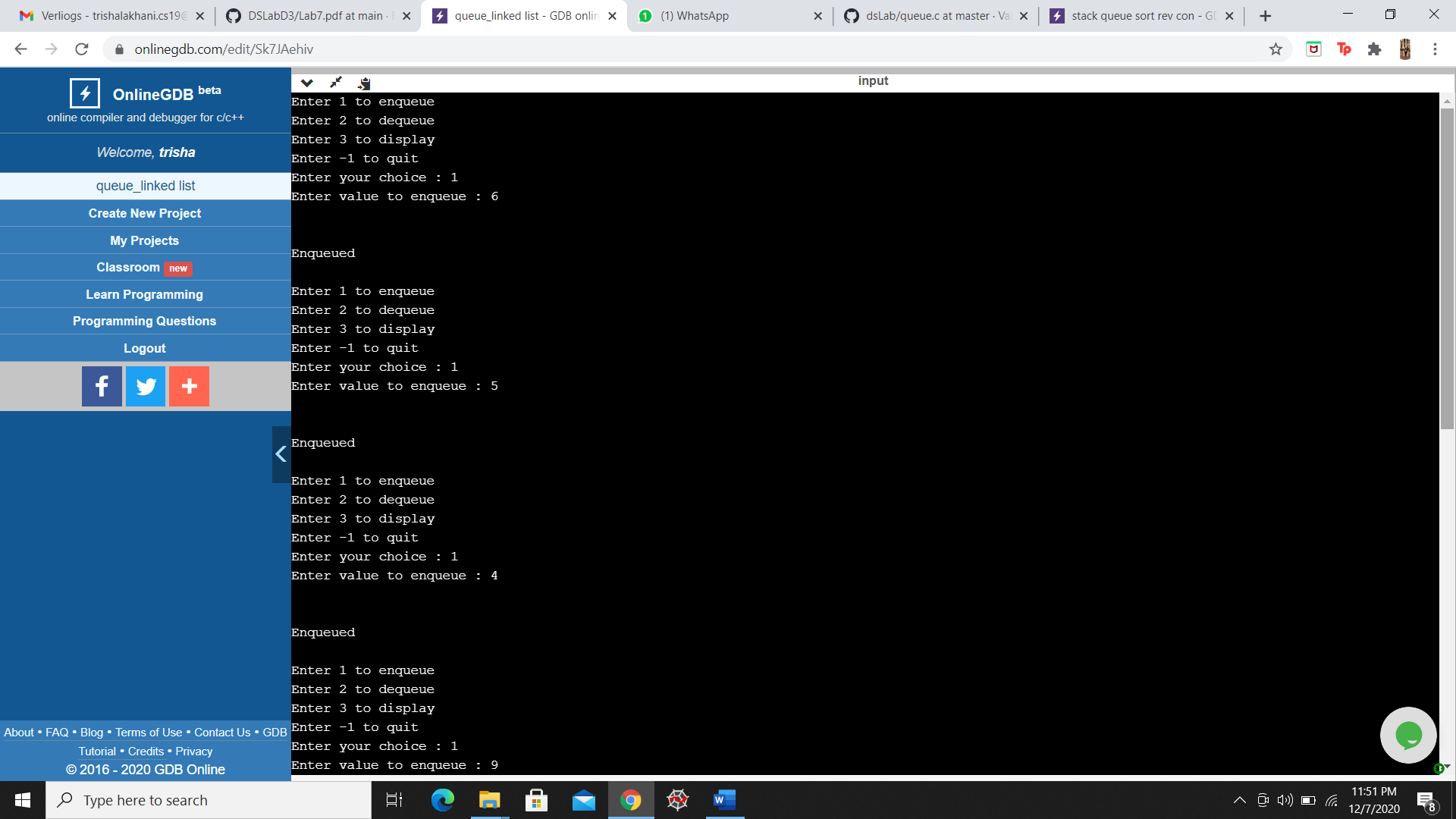
}

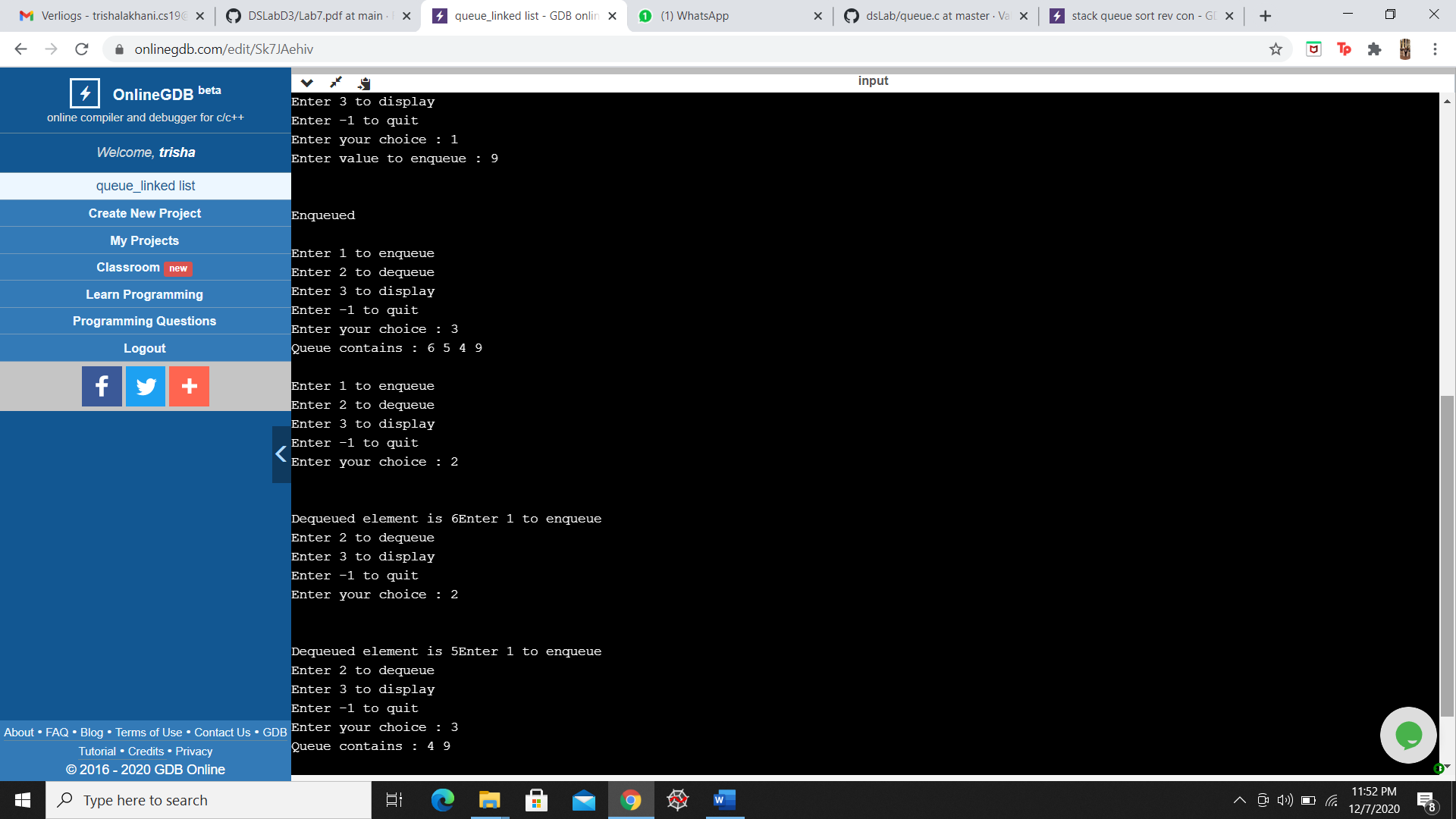
}

return 0;

}

**OUTPUT**

****

****